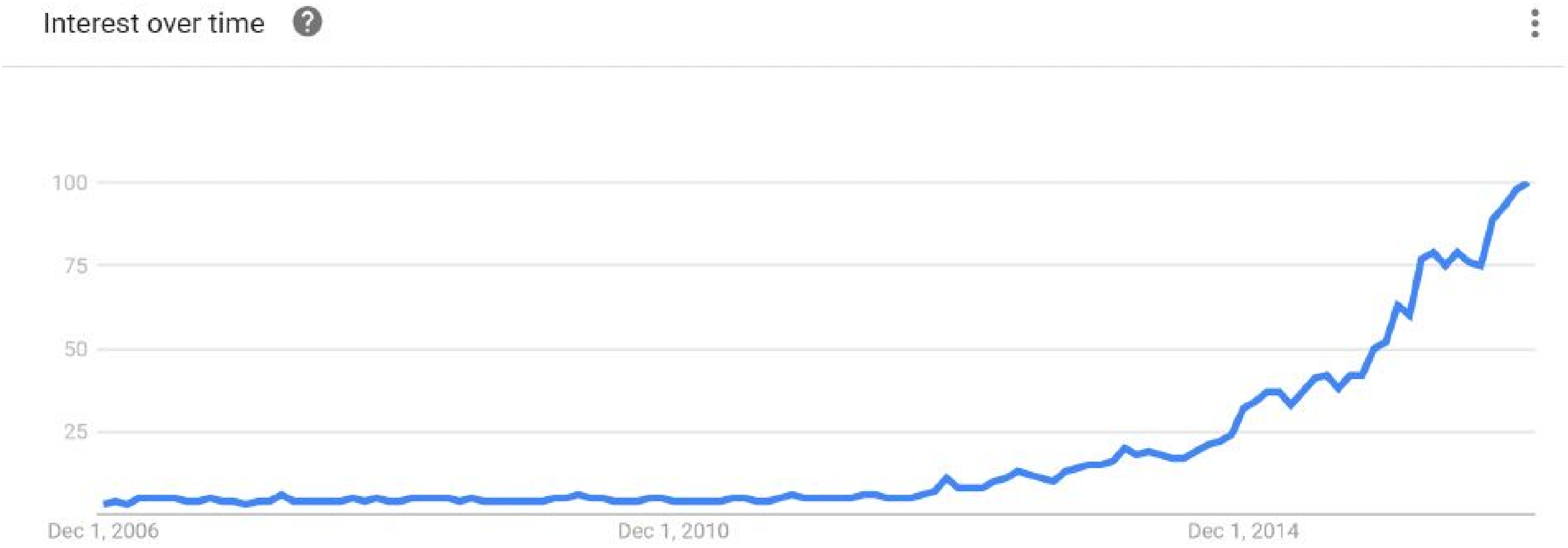
DATA SCIENCE RETREAT®

Deep learning basis

Jose Quesada DSR 09
Jan 2017


Interest over time
100
75
50
25
Dec 1, 2006
Dec 1, 2010
Dec 1, 2014

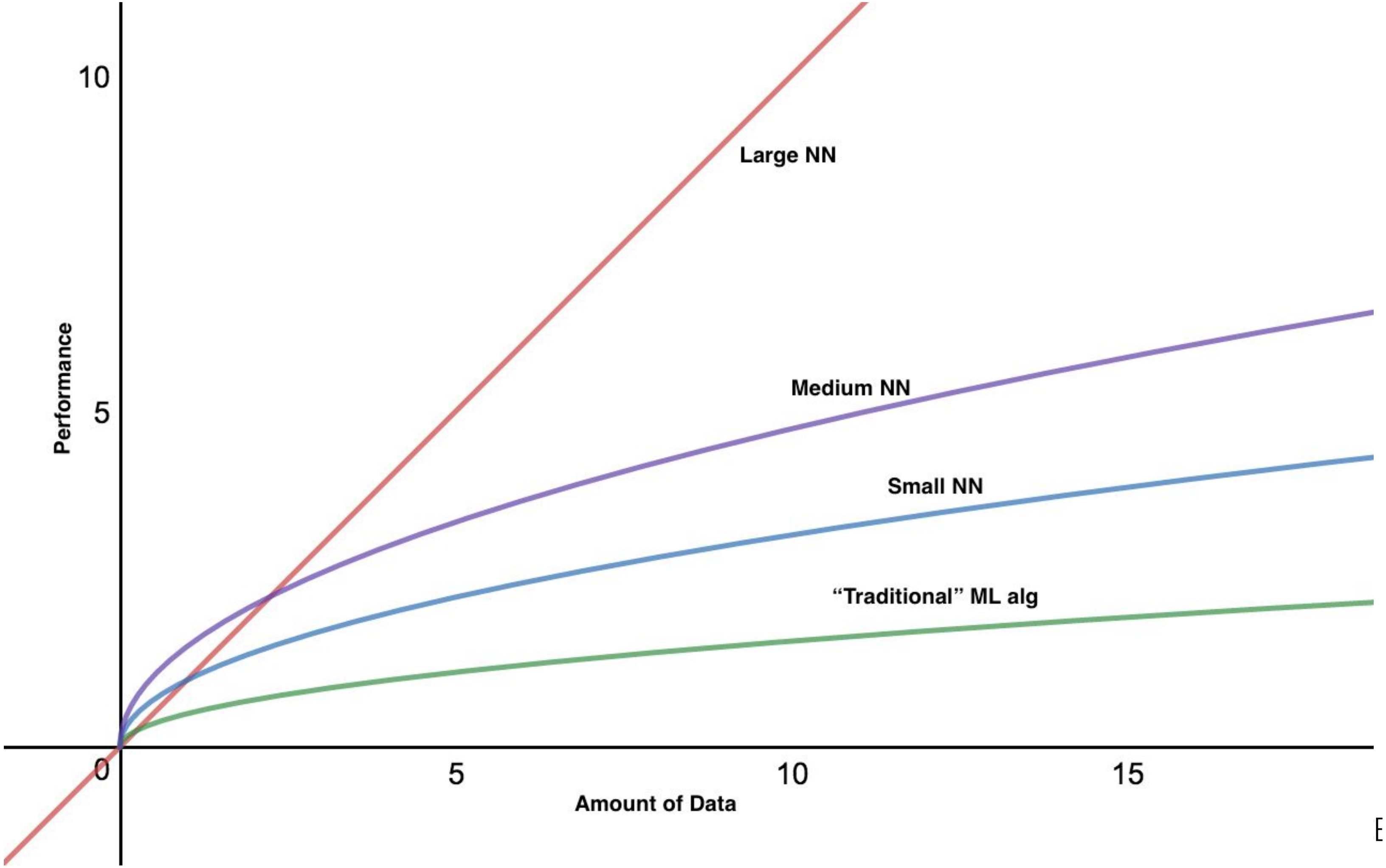
Economist.com

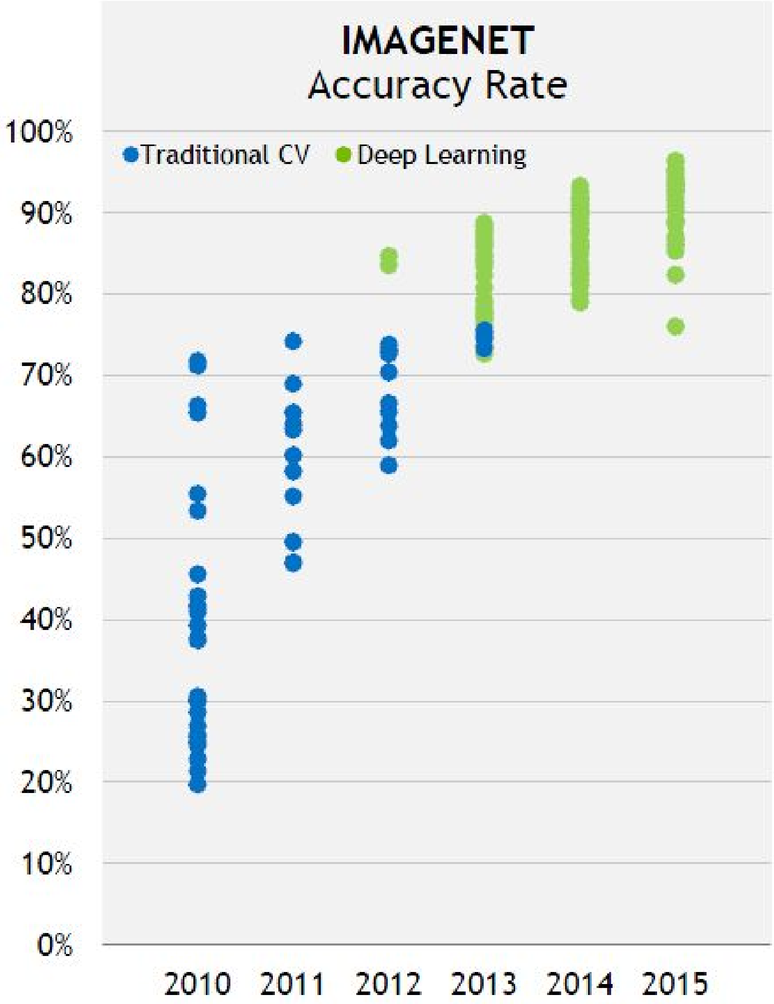
DATA SCIENCE RETREAT®


Algorithms took your jobs,
not immigrants.

The one second rule

Anything that takes a person less than one second of thought we can automate using AI either now or in the future


10
5
0
Performance
Large NN
Medium NN
Small NN
"Traditional" ML alg
5
10
15
Amount of Data

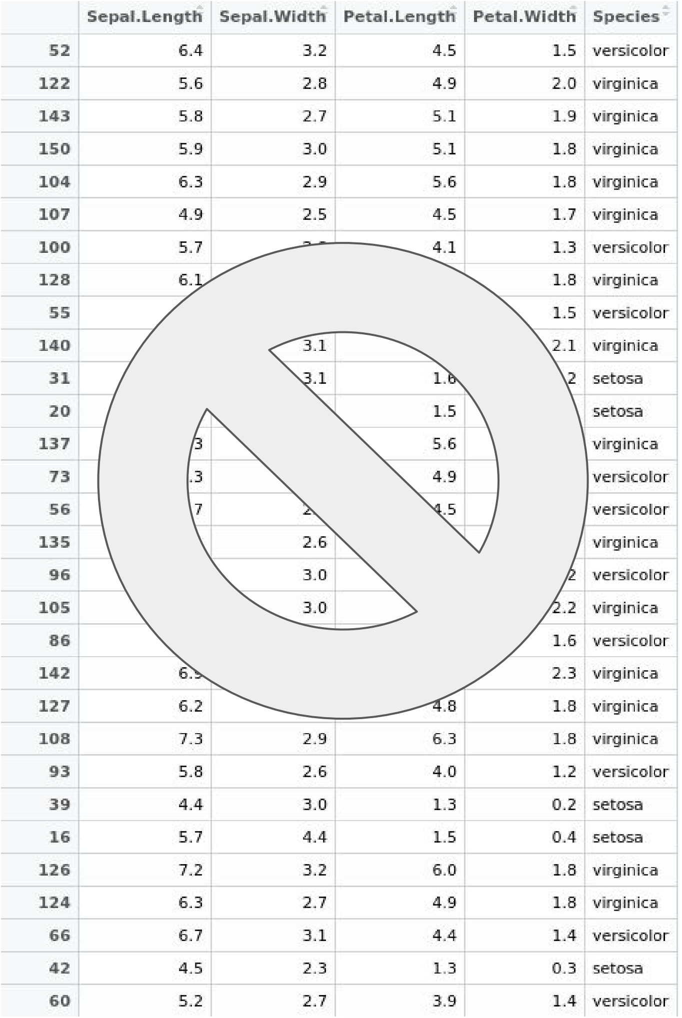


DATA SCIENCE RETREAT®

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
52	6.4	3.2	4.5	1.5	versicolor
122	5.6	2.8	4.9	2.0	virginica
143	5.8	2.7	5.1	1.9	virginica
150	5.9	3.0	5.1	1.8	virginica
104	6.3	2.9	5.6	1.8	virginica
107	4.9	2.5	4.5	1.7	virginica
100	5.7	2.8	4.1	1.3	versicolor
128	6.1	3.0	4.9	1.8	virginica
55	6.5	2.8	4.6	1.5	versicolor
140	6.9	3.1	5.4	2.1	virginica
31	4.8	3.1	1.6	0.2	setosa
20	5.1	3.8	1.5	0.3	setosa
137	6.3	3.4	5.6	2.4	virginica
73	6.3	2.5	4.9	1.5	versicolor
56	5.7	2.8	4.5	1.3	versicolor
135	6.1	2.6	5.6	1.4	virginica
96	5.7	3.0	4.2	1.2	versicolor
105	6.5	3.0	5.8	2.2	virginica
86	6.0	3.4	4.5	1.6	versicolor
142	6.9	3.1	5.1	2.3	virginica
127	6.2	2.8	4.8	1.8	virginica
108	7.3	2.9	6.3	1.8	virginica
93	5.8	2.6	4.0	1.2	versicolor
39	4.4	3.0	1.3	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
126	7.2	3.2	6.0	1.8	virginica
124	6.3	2.7	4.9	1.8	virginica
66	6.7	3.1	4.4	1.4	versicolor
42	4.5	2.3	1.3	0.3	setosa
60	5.2	2.7	3.9	1.4	versicolor

→ Machine learning → Predictions

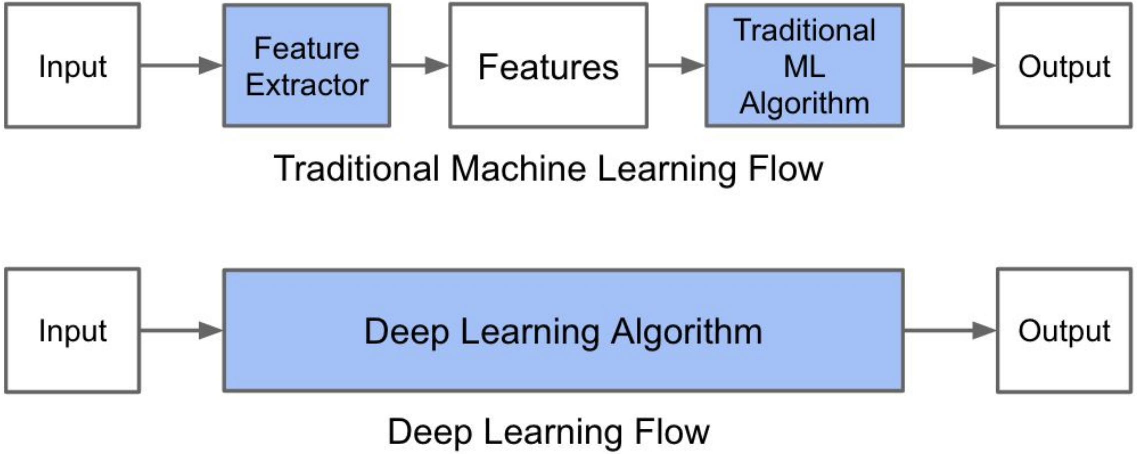
Deep learning solves a central problem in ‘representation’: it learns representations that are expressed in terms of other, simpler representations



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
52	6.4	3.2	4.5	1.5	versicolor
122	5.6	2.8	4.9	2.0	virginica
143	5.8	2.7	5.1	1.9	virginica
150	5.9	3.0	5.1	1.8	virginica
104	6.3	2.9	5.6	1.8	virginica
107	4.9	2.5	4.5	1.7	virginica
100	5.7	[illegible]	4.1	1.3	versicolor
128	6.1	[illegible]	[illegible]	1.8	virginica
55	[illegible]	[illegible]	[illegible]	1.5	versicolor
140	[illegible]	3.1	[illegible]	2.1	virginica
31	[illegible]	3.1	1.6	[illegible]	setosa
20	[illegible]	[illegible]	1.5	[illegible]	setosa
137	[illegible]	[illegible]	5.6	[illegible]	virginica
73	[illegible]	[illegible]	4.9	[illegible]	versicolor
56	[illegible]	[illegible]	[illegible]	[illegible]	versicolor
135	[illegible]	2.6	[illegible]	[illegible]	virginica
96	[illegible]	3.0	[illegible]	[illegible]	versicolor
105	[illegible]	3.0	[illegible]	2.2	virginica
86	[illegible]	[illegible]	[illegible]	1.6	versicolor
142	[illegible]	[illegible]	[illegible]	2.3	virginica
127	6.2	[illegible]	4.8	1.8	virginica
108	7.3	2.9	6.3	1.8	virginica
93	5.8	2.6	4.0	1.2	versicolor
39	4.4	3.0	1.3	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
126	7.2	3.2	6.0	1.8	virginica
124	6.3	2.7	4.9	1.8	virginica
66	6.7	3.1	4.4	1.4	versicolor
42	4.5	2.3	1.3	0.3	setosa
60	5.2	2.7	3.9	1.4	versicolor

Machine learning

Predictions

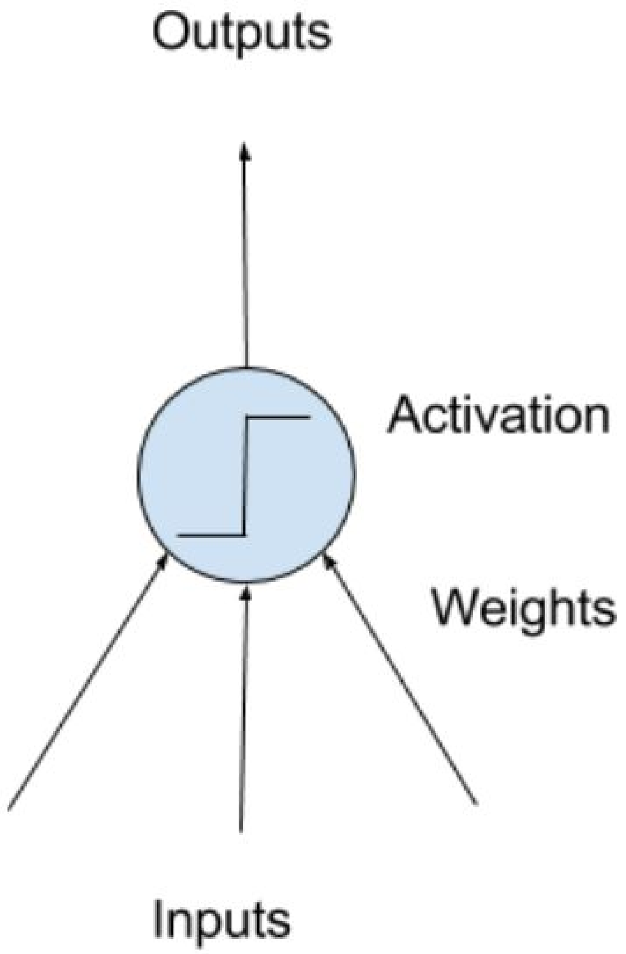


DATA SCIENCE RETREAT®

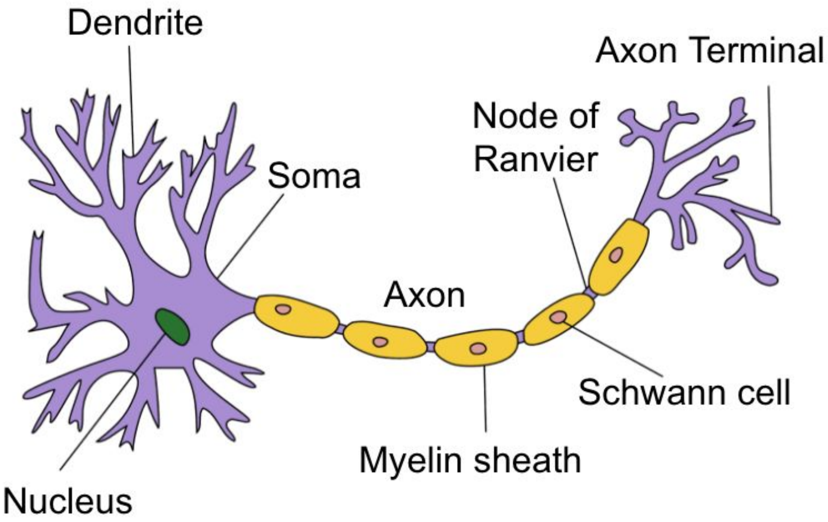
The end of feature engineering?

And the beginning of architecture engineering?

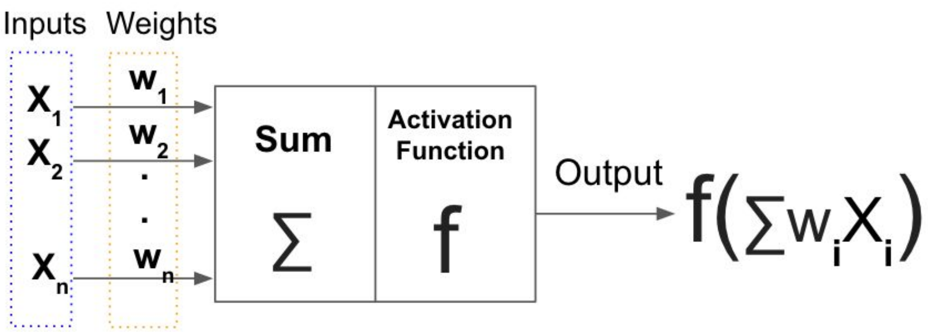
Neurons



What is a neuron, and how do you represent it with math?

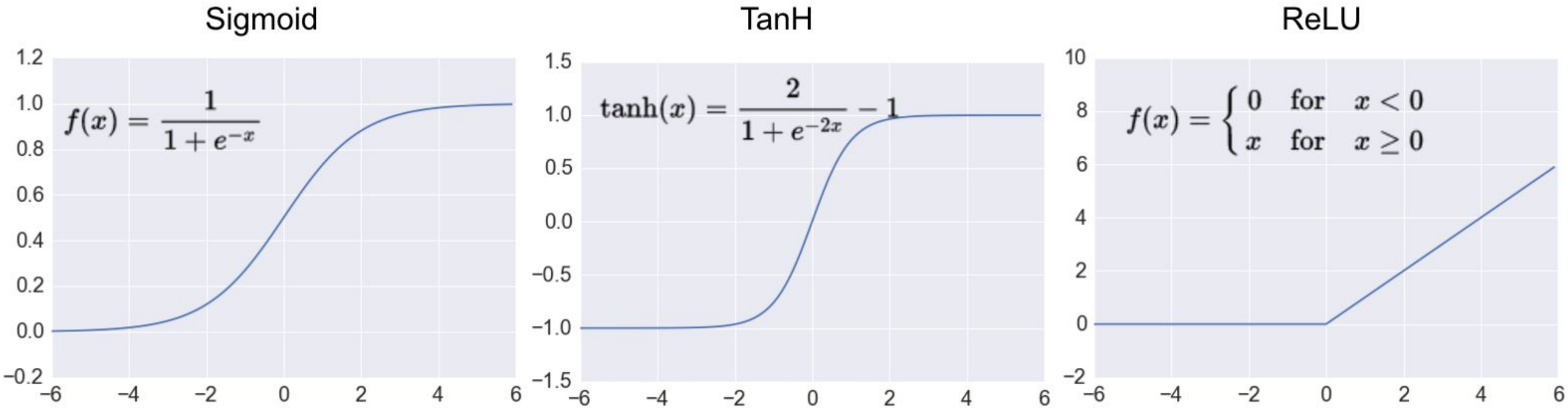


Structure of a typical neuron
(source: Wikipedia)

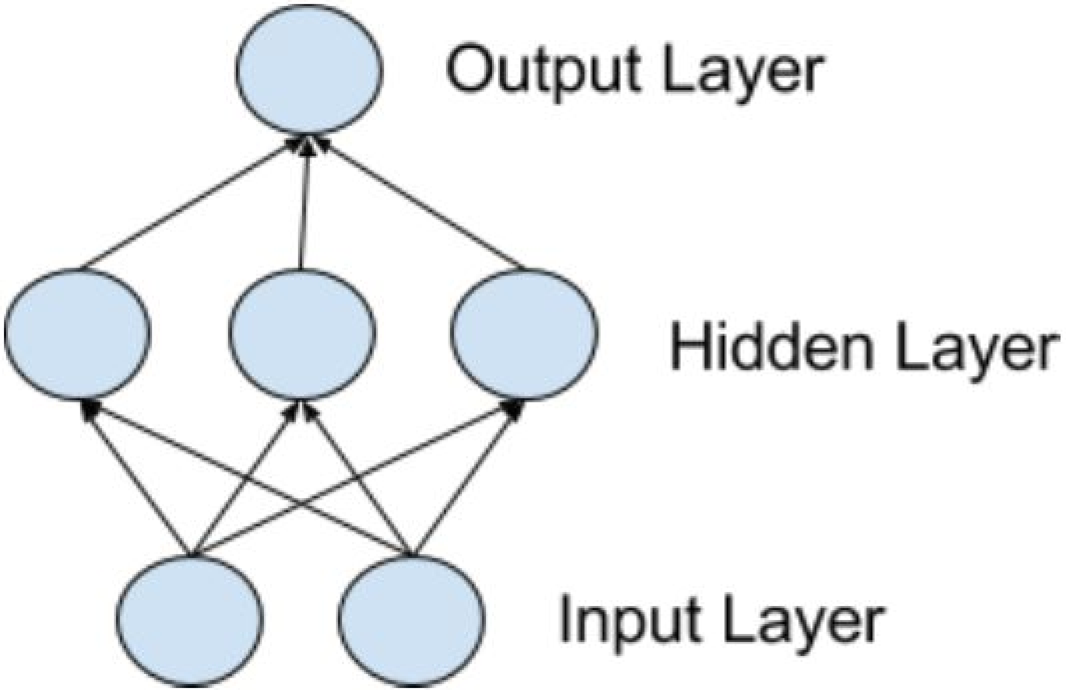


Structure of artificial neuron

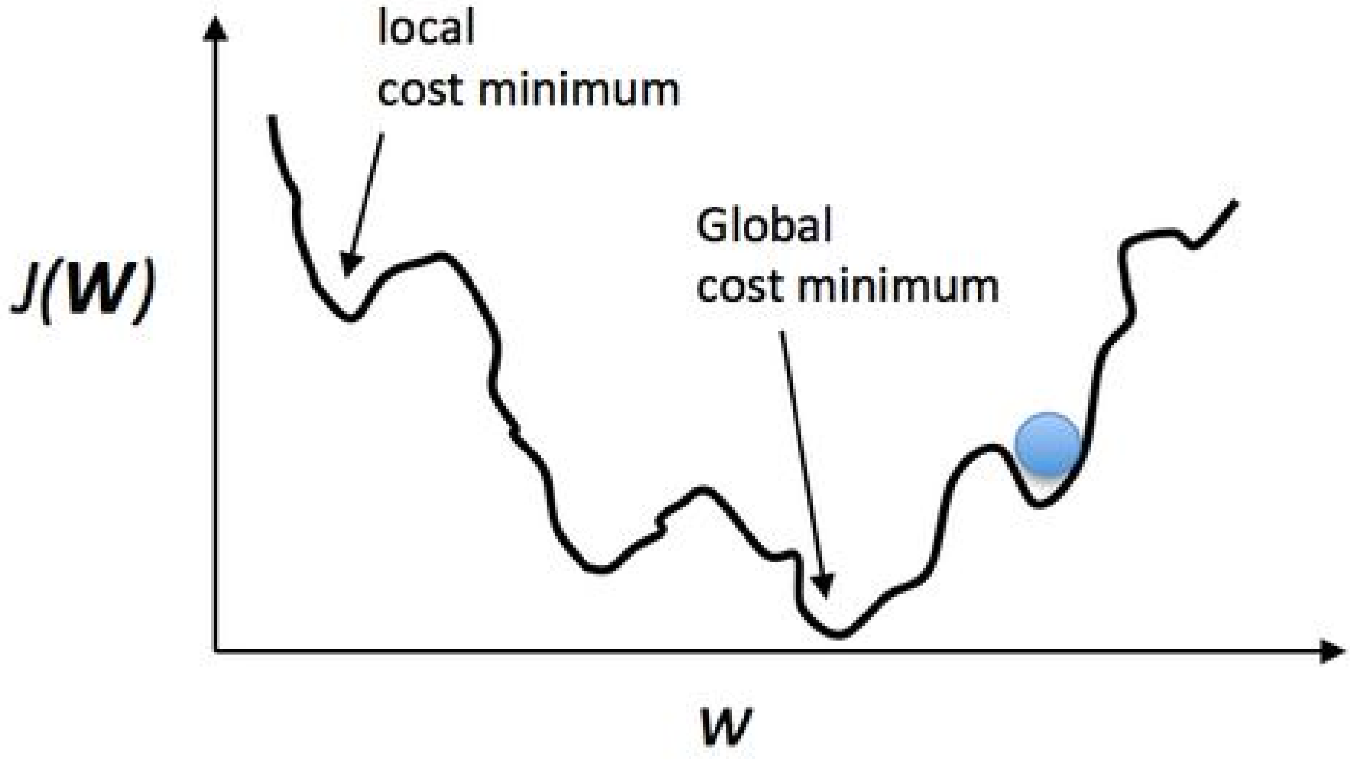
Activation functions



Networks



Finding minima

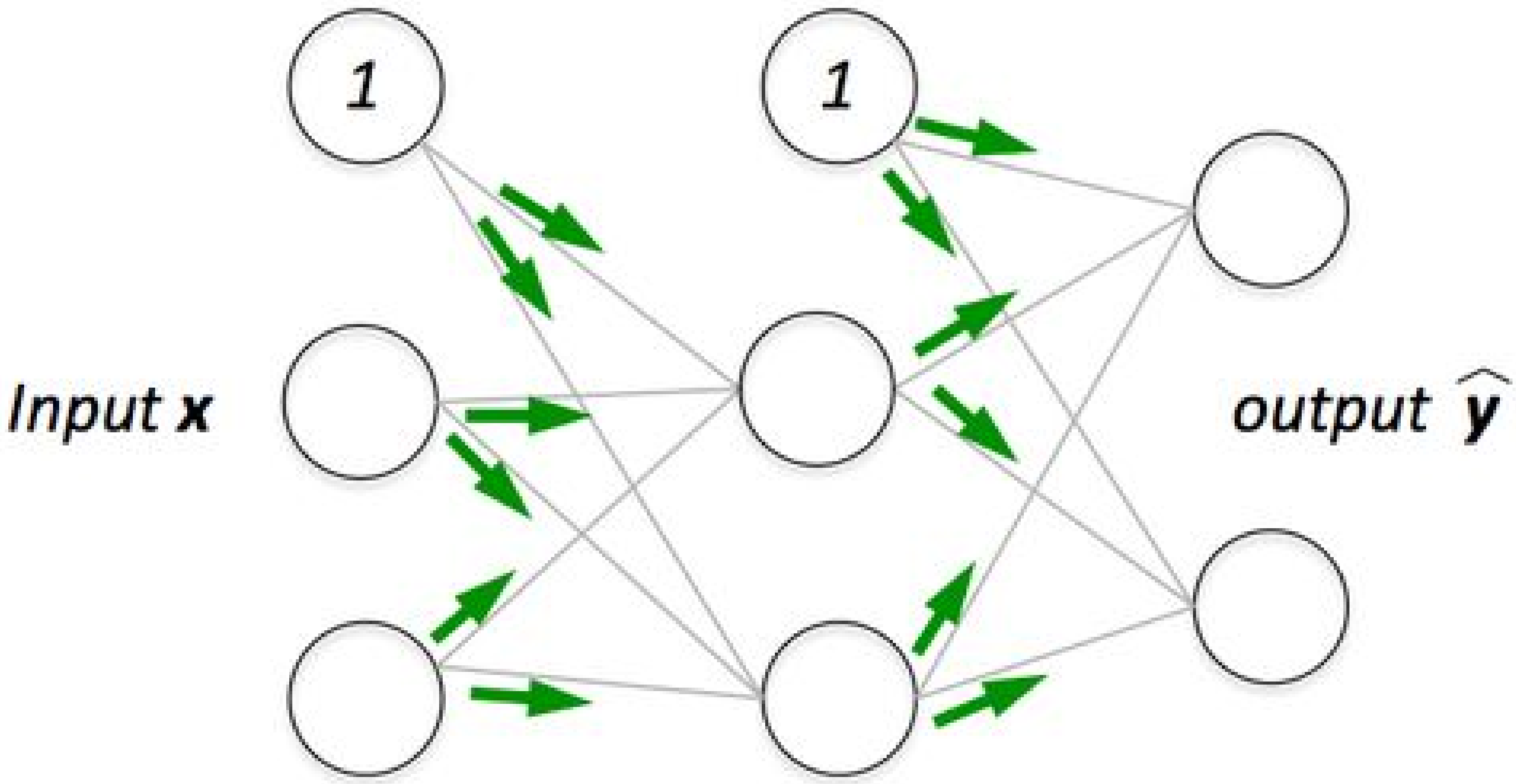


ATA SCIENCE RETREAT®

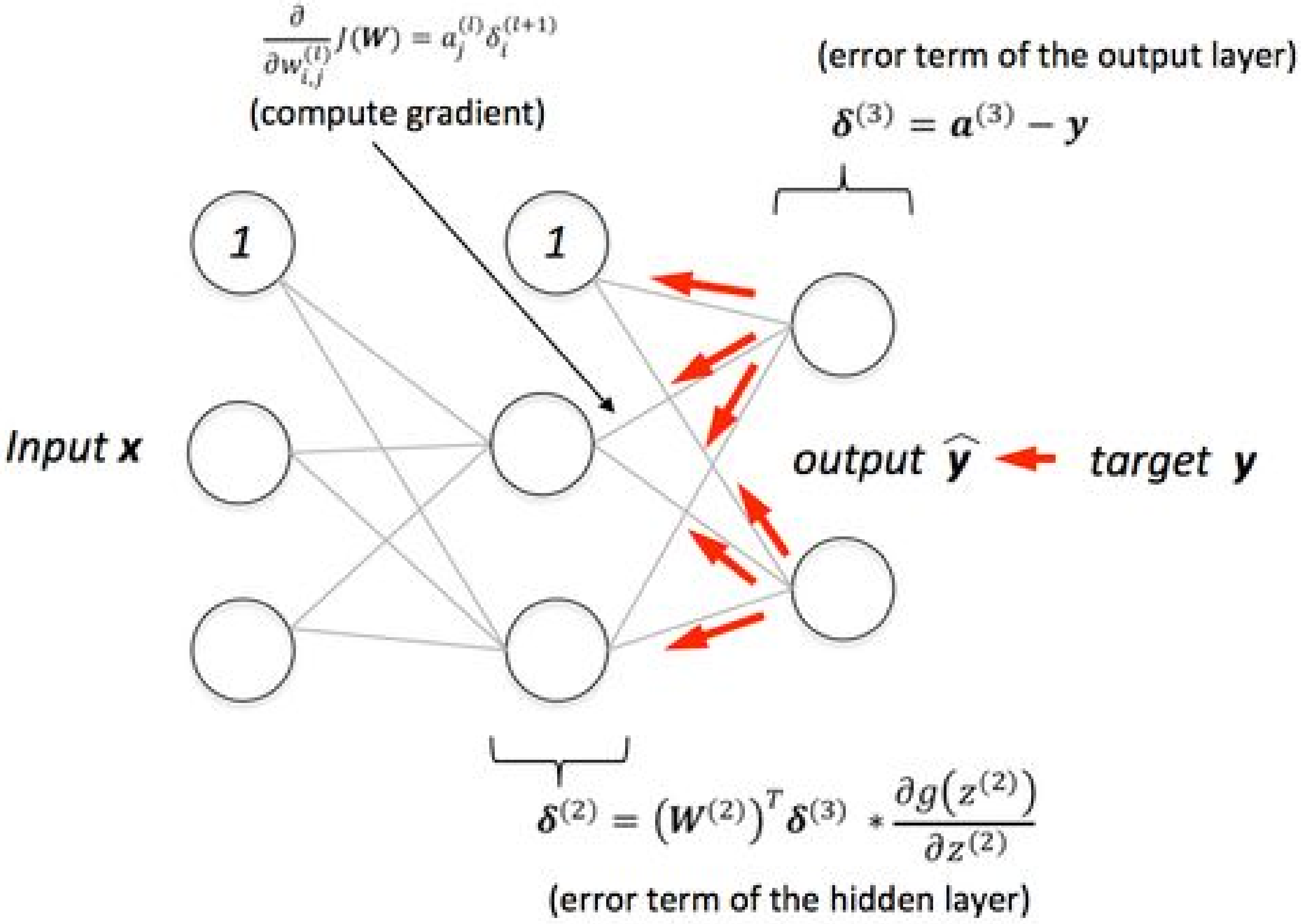
Backpropagation

It's an algorithm that, given the error in the output of the network, tries to determine how to distribute it to the weights (determining how much each weight is responsible for the final error), so that you know how to update the weights

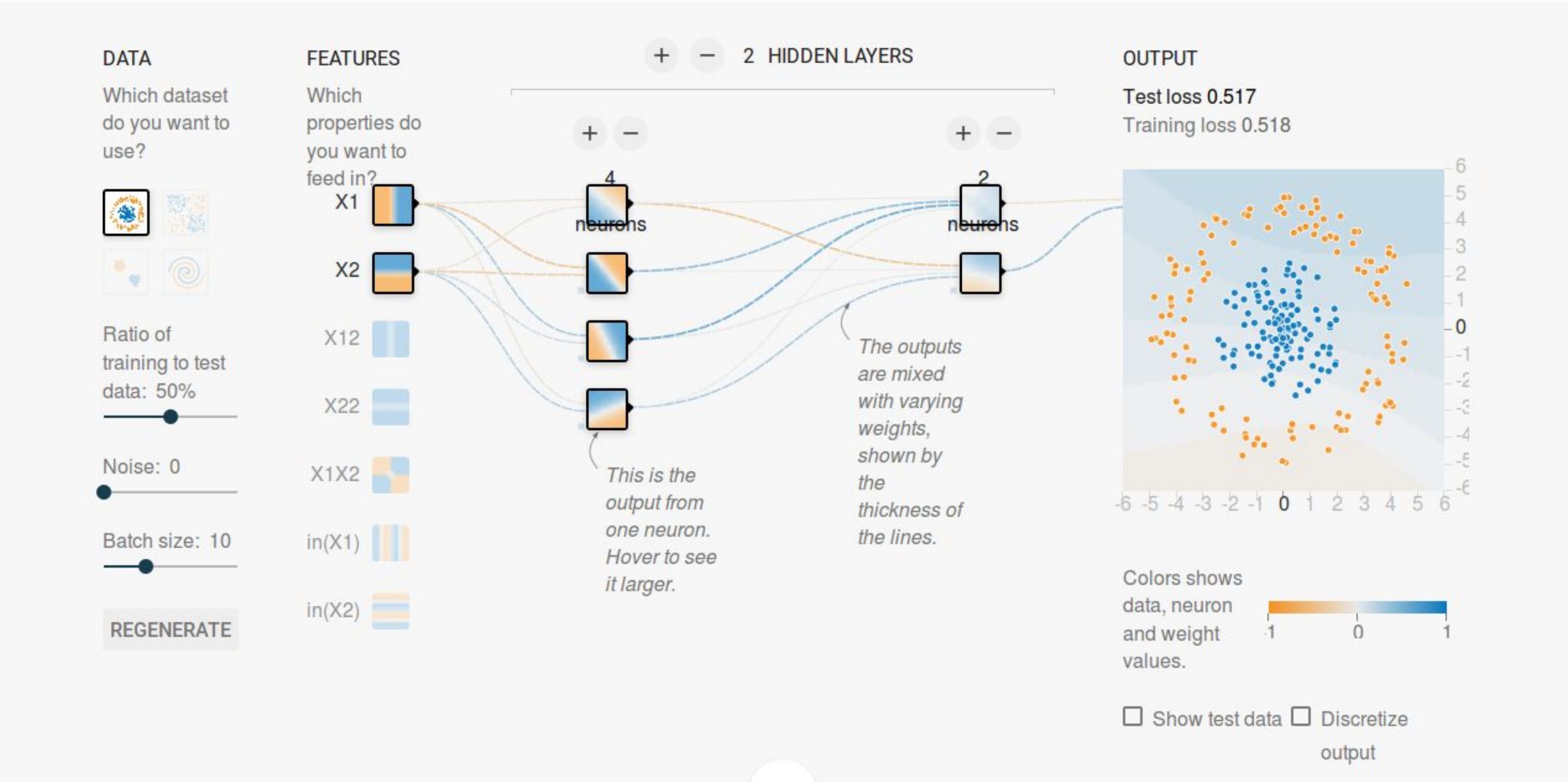
Based on Spreading activation, vector by matrix multiplication



AT®



$\frac{\partial}{\partial w_{i,j}^{(l)}} J(W) = a_j^{(l)} \delta_i^{(l+1)}$
(compute gradient)
(error term of the output layer)
$\delta^{(3)} = a^{(3)} - y$
1
1
Input x
output $\hat{y}$
target y
$\delta^{(2)} = (W^{(2)})^T \delta^{(3)} * \frac{\partial g(z^{(2)})}{\partial z^{(2)}}$
(error term of the hidden layer)



http://playground.tensorflow.org Based on Andrej Karpathy's convnet.js demo

DATA SCIENCE RETREAT®


a row of bikes parked next to each other

ETREAT®

GPUs vs CPUs
(or why do we need nvidia?)

DATA SCIENCE RETREAT®

GPUs are designed with one goal in mind: process graphics really fast. Since this is the only concern they have, there have been some specialized optimizations in place that allow for certain calculations to go a LOT faster than they would in a traditional processor.

Most deep learning libraries - and in fact most software applications in general - just use a single thread. This means that multi-core CPUs are rather useless.

		Desktop CPU				Server CPU						GPU		
		1	2	4	8	1	2	4	8	16	32	G.980	G.1080	T.K80
FCN-5	Caffe	0.919	0.495	**0.480**	-	0.769	0.446	0.354	0.269	**0.287**	0.688	0.020	**0.017**	0.028
	CNTK	2.351	1.239	0.961	**0.810**	2.311	1.229	0.827	0.546	**0.530**	0.549	0.043	**0.033**	0.052
	TF	7.205	4.904	2.626	**1.933**	7.449	5.203	2.803	1.574	0.857	**0.594**	0.071	**0.063**	0.098
	Torch	1.227	0.655	**0.661**	-	1.030	0.740	0.535	0.440	**0.425**	0.892	0.044	**0.039**	0.056
FCN-8	Caffe	1.035	0.857	**0.572**	-	0.888	0.613	0.391	0.319	**0.316**	0.810	0.023	**0.019**	0.033
	CNTK	2.641	1.402	1.393	**0.919**	2.514	1.391	0.884	0.633	**0.579**	0.653	0.048	**0.037**	0.059
	TF	7.166	4.863	2.629	**1.955**	7.759	5.198	2.896	1.577	0.891	**0.619**	0.074	**0.065**	0.106
	Torch	1.316	0.706	**0.448**	0.881	1.106	0.774	0.559	0.475	**0.443**	0.975	0.046	**0.046**	0.057
AlexNet	Caffe	2.507	1.492	**1.005**	1.460	1.917	1.281	0.975	0.996	**1.035**	1.239	0.042	**0.038**	0.089
	CNTK	6.661	3.556	**2.123**	4.232	6.716	3.966	2.618	1.987	**1.446**	1.578	0.052	**0.043**	0.089
	TF	3.192	2.219	1.346	**1.134**	3.720	2.671	1.416	0.812	**0.516**	0.627	0.037	**0.012**	0.064
	Torch	4.689	2.473	**2.090**	4.012	3.293	1.883	1.156	1.145	**1.083**	1.182	0.036	**0.034**	0.073
ResNet	Caffe	7.810	5.312	**4.056**	5.876	6.150	5.390	4.314	**4.124**	4.500	5.034	-	**0.208**	0.353
	CNTK	-	-	-	-	-	-	-	-	-	-	0.289	**0.261**	0.468
	TF	21.63	12.19	7.655	**6.340**	20.49	14.340	7.703	4.600	**2.890**	3.937	0.226	**0.085**	0.392
	Torch	12.10	7.147	**-**	-	10.16	6.928	4.856	3.757	**3.524**	4.165	0.216	**0.181**	0.412
LSTM-32	CNTK	0.579	0.391	**0.306**	1.153	0.591	0.418	0.353	0.338	**0.342**	0.442	0.433	**0.366**	0.602
	TF	9.305	3.432	**2.020**	1.722	6.453	3.782	2.167	1.228	**0.769**	0.706	0.086	**0.083**	0.122
	Torch	4.872	2.680	**2.366**	3.645	4.704	2.971	2.067	1.706	**1.763**	2.900	0.124	**0.098**	0.204
LSTM-64	CNTK	1.026	0.690	**0.535**	1.860	1.043	0.756	0.622	0.585	**0.648**	0.790	0.779	**0.649**	1.052
	TF	11.69	7.292	3.515	**3.476**	12.76	7.823	4.402	2.524	1.590	**1.469**	0.178	**0.173**	0.233
	Torch	9.622	5.323	**4.980**	6.975	9.364	5.613	4.054	**3.252**	3.357	5.815	0.247	**0.194**	0.406

Source: ‘Deep learning with Python’, Jason Brownlee (2017) Online self-published book, with author’s permission

Multilayer perceptron (simple starting point)

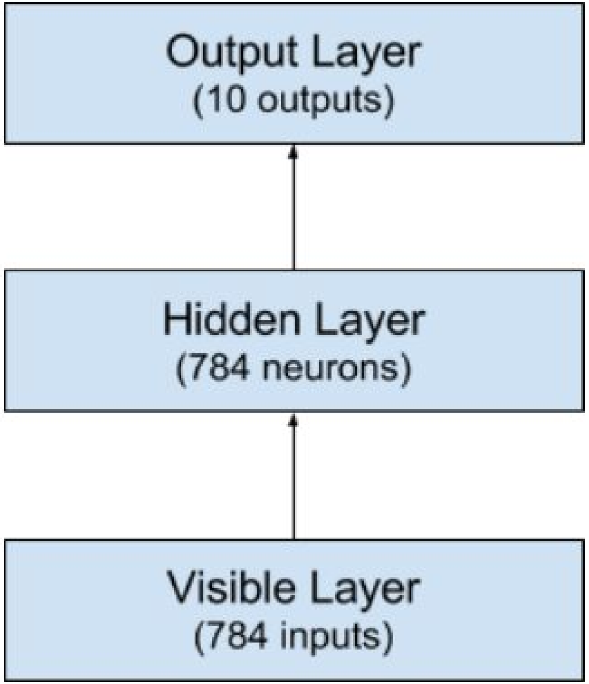
Quiz time. Recreating classic models with NN

A regression problem

A binary classification problem

A multiclass classification problem

MNIST multilayer perceptron in Keras



```
python mnist_mlp_baseline.py
Using TensorFlow backend.
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
4s - loss: 0.2836 - acc: 0.9189 - val_loss: 0.1400 - val_acc: 0.9593
Epoch 2/10
4s - loss: 0.1123 - acc: 0.9670 - val_loss: 0.0932 - val_acc: 0.9718
Epoch 3/10
4s - loss: 0.0724 - acc: 0.9790 - val_loss: 0.0781 - val_acc: 0.9767
Epoch 4/10
4s - loss: 0.0511 - acc: 0.9854 - val_loss: 0.0733 - val_acc: 0.9775
Epoch 5/10
4s - loss: 0.0376 - acc: 0.9896 - val_loss: 0.0689 - val_acc: 0.9783
Epoch 6/10
4s - loss: 0.0271 - acc: 0.9925 - val_loss: 0.0643 - val_acc: 0.9803
Epoch 7/10
5s - loss: 0.0206 - acc: 0.9949 - val_loss: 0.0588 - val_acc: 0.9818
Epoch 8/10
5s - loss: 0.0135 - acc: 0.9974 - val_loss: 0.0595 - val_acc: 0.9814
Epoch 9/10
5s - loss: 0.0110 - acc: 0.9977 - val_loss: 0.0559 - val_acc: 0.9817
Epoch 10/10
5s - loss: 0.0083 - acc: 0.9985 - val_loss: 0.0581 - val_acc: 0.9825
```

Baseline Error: 1.75%

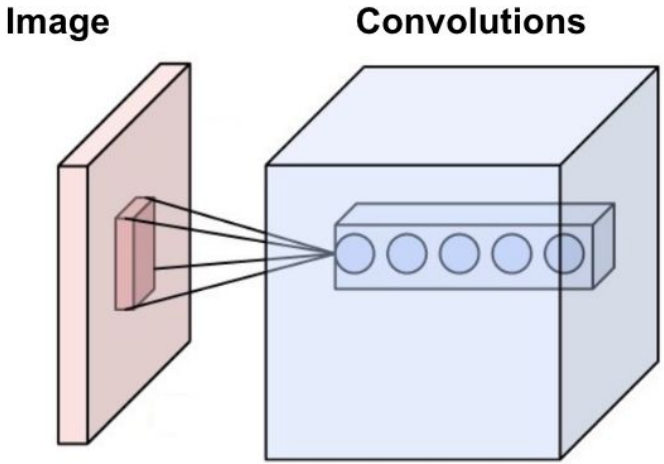
Convolutional NN


Real data scientists stay sexy by eating world famous
Convolutional
Neural
Nachos®
Cognitive Foods International Inc.

There are three types of layers in a Convolutional Neural Network:

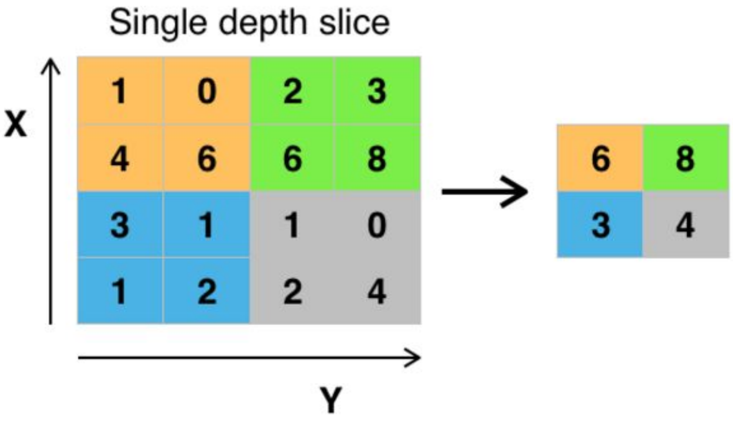
1. Convolutional Layers
2. Pooling Layers
3. Fully-Connected Layers

1. Convolutional Layers



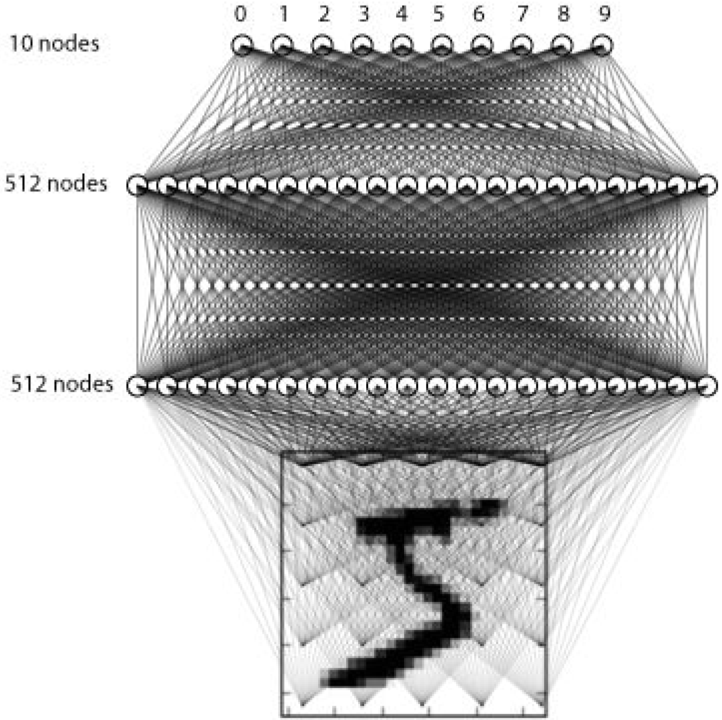
Neurons of a convolutional layer, connected to their receptive field
(source: Wikipedia)

2. Pooling Layers

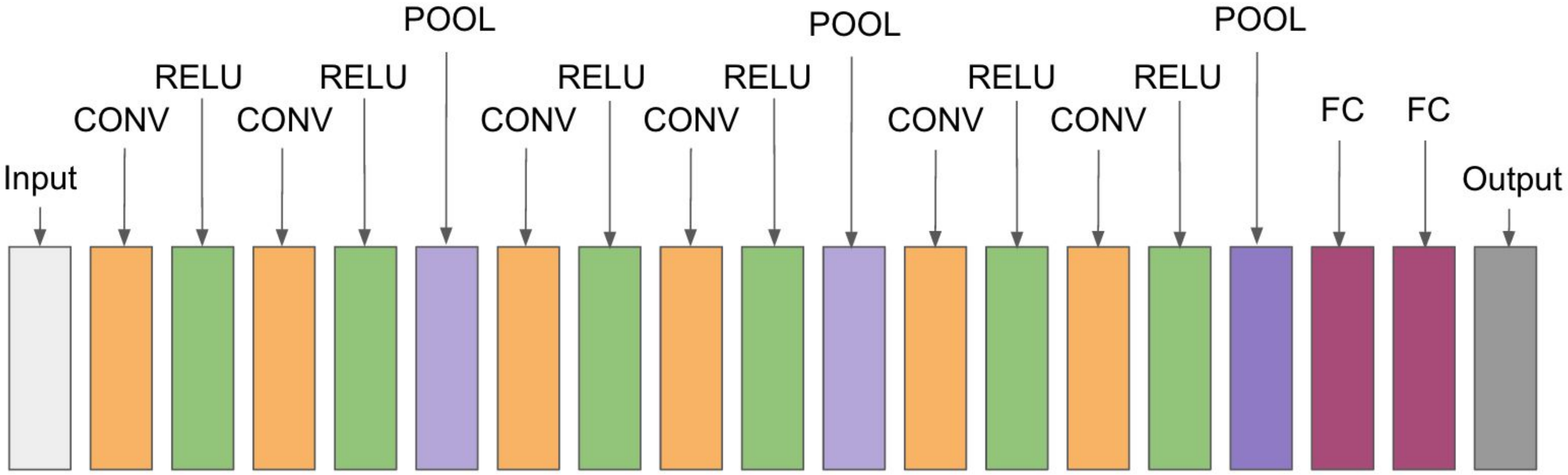


Max pooling with a 2x2 filter and stride = 2
(source: Wikipedia)

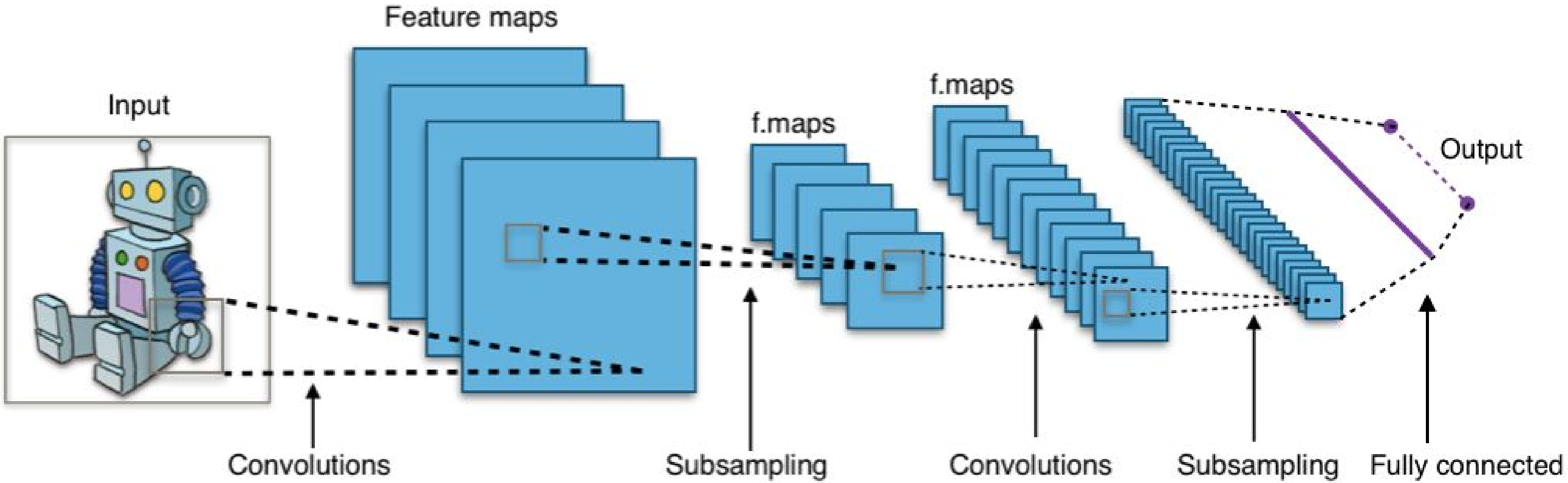
3. Fully-Connected Layers



They combine into an ‘architecture’



Example of a CNN Architecture


Feature maps
Input
f.maps
f.maps
Output
Convolutions
Subsampling
Convolutions
Subsampling
Fully connected

What can we change?

- Regularization
- Learning rate

Minibatch size

- Augmentation
- More convolutions filters
- Dropout
- Momentum
- Better Optimizer

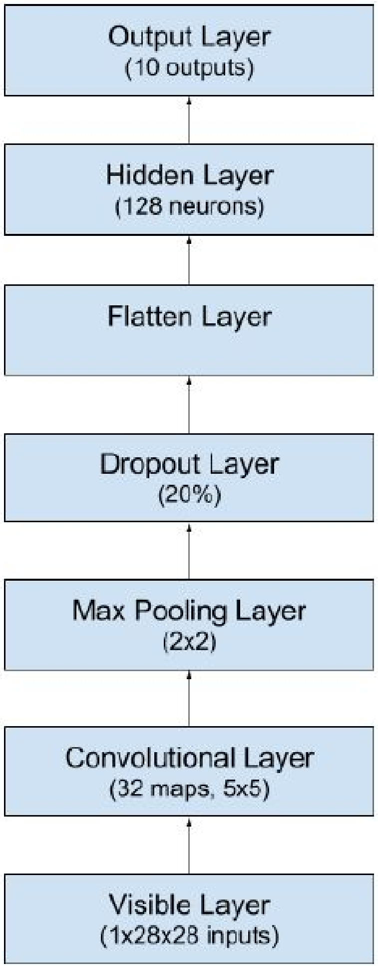
1. ReLUs (Or other state of the art activation functions)
2. Learning rate •
3. Mini batch Size
4. Momentum / Better Optimizer

And only if the system is learning:

5. Regularization
6. More convolutions filters
7. Dropout
8. Augmentation

DATA SCIENCE RETREAT®

Exercise: Deep net Convnet for mnist, sample architecture



```
python mnist_cnn.py
Using TensorFlow backend.
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
88s - loss: 0.2537 - acc: 0.9279 - val_loss: 0.0854 - val_acc: 0.9751
Epoch 2/10
97s - loss: 0.0766 - acc: 0.9773 - val_loss: 0.0596 - val_acc: 0.9803
Epoch 3/10
99s - loss: 0.0544 - acc: 0.9832 - val_loss: 0.0451 - val_acc: 0.9861
Epoch 4/10
101s - loss: 0.0423 - acc: 0.9873 - val_loss: 0.0371 - val_acc: 0.9880
Epoch 5/10
101s - loss: 0.0340 - acc: 0.9895 - val_loss: 0.0389 - val_acc: 0.9883
Epoch 6/10
100s - loss: 0.0286 - acc: 0.9910 - val_loss: 0.0317 - val_acc: 0.9901
Epoch 7/10
99s - loss: 0.0234 - acc: 0.9927 - val_loss: 0.0327 - val_acc: 0.9895
Epoch 8/10
99s - loss: 0.0189 - acc: 0.9938 - val_loss: 0.0321 - val_acc: 0.9899
Epoch 9/10
99s - loss: 0.0170 - acc: 0.9944 - val_loss: 0.0345 - val_acc: 0.9901
Epoch 10/10
100s - loss: 0.0143 - acc: 0.9957 - val_loss: 0.0305 - val_acc: 0.9913
```

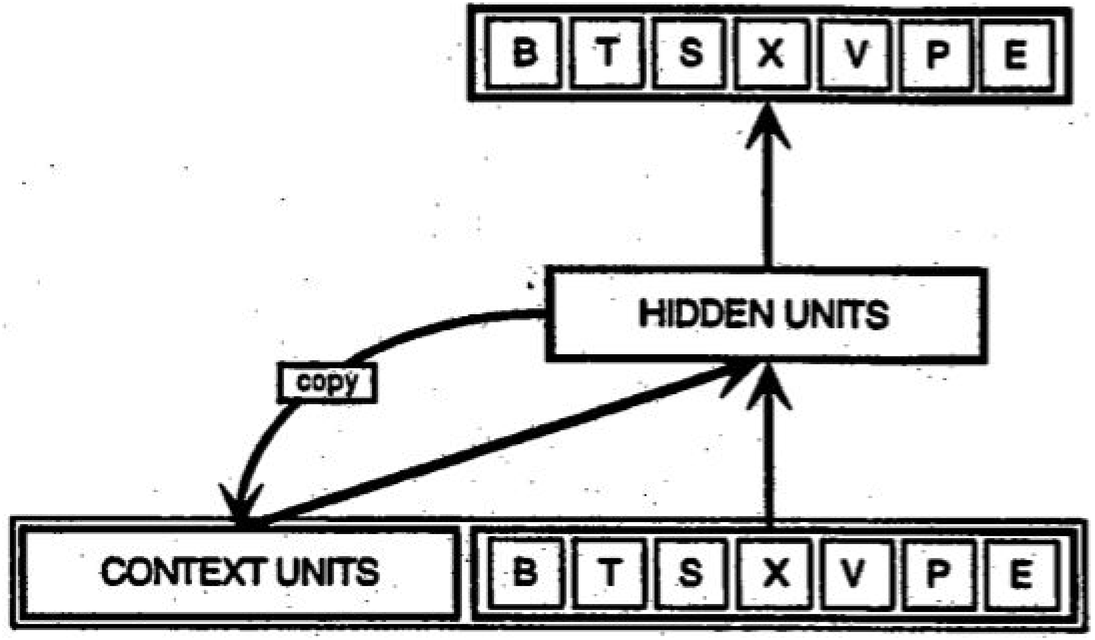
CNN Error: 0.87%

(optional) Hands-on time, imdb sentiment analysis

LSTM

RNN and LTSM networks

Recurrent Networks



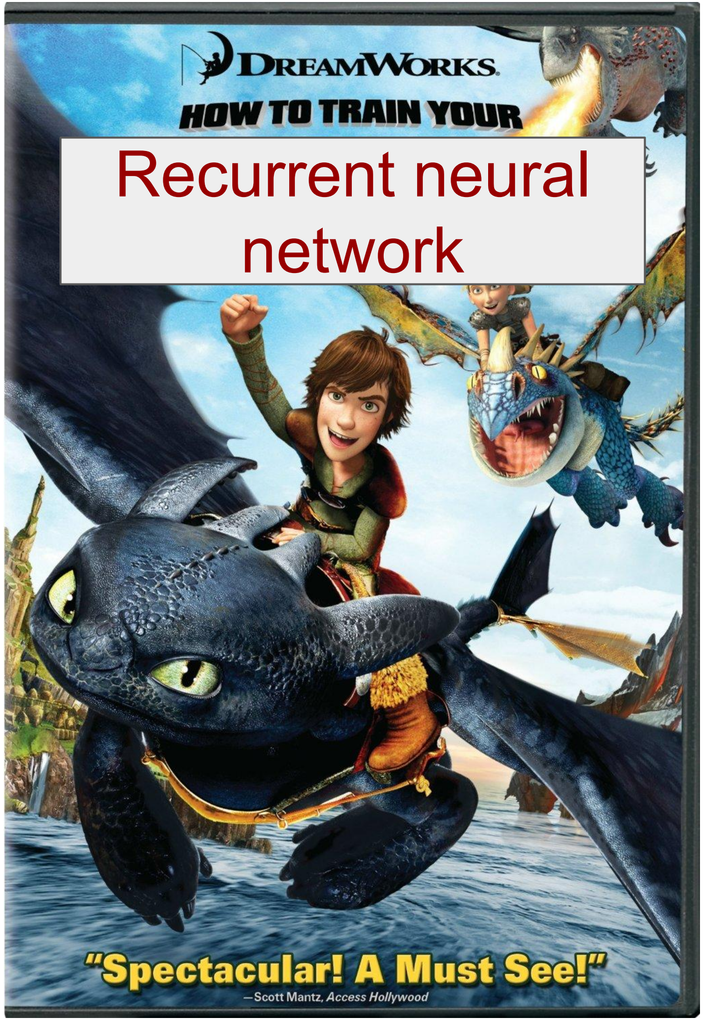
https://web.stanford.edu/group/pdplab/pdphandbook/handbookch8.html ,
Elman 1990

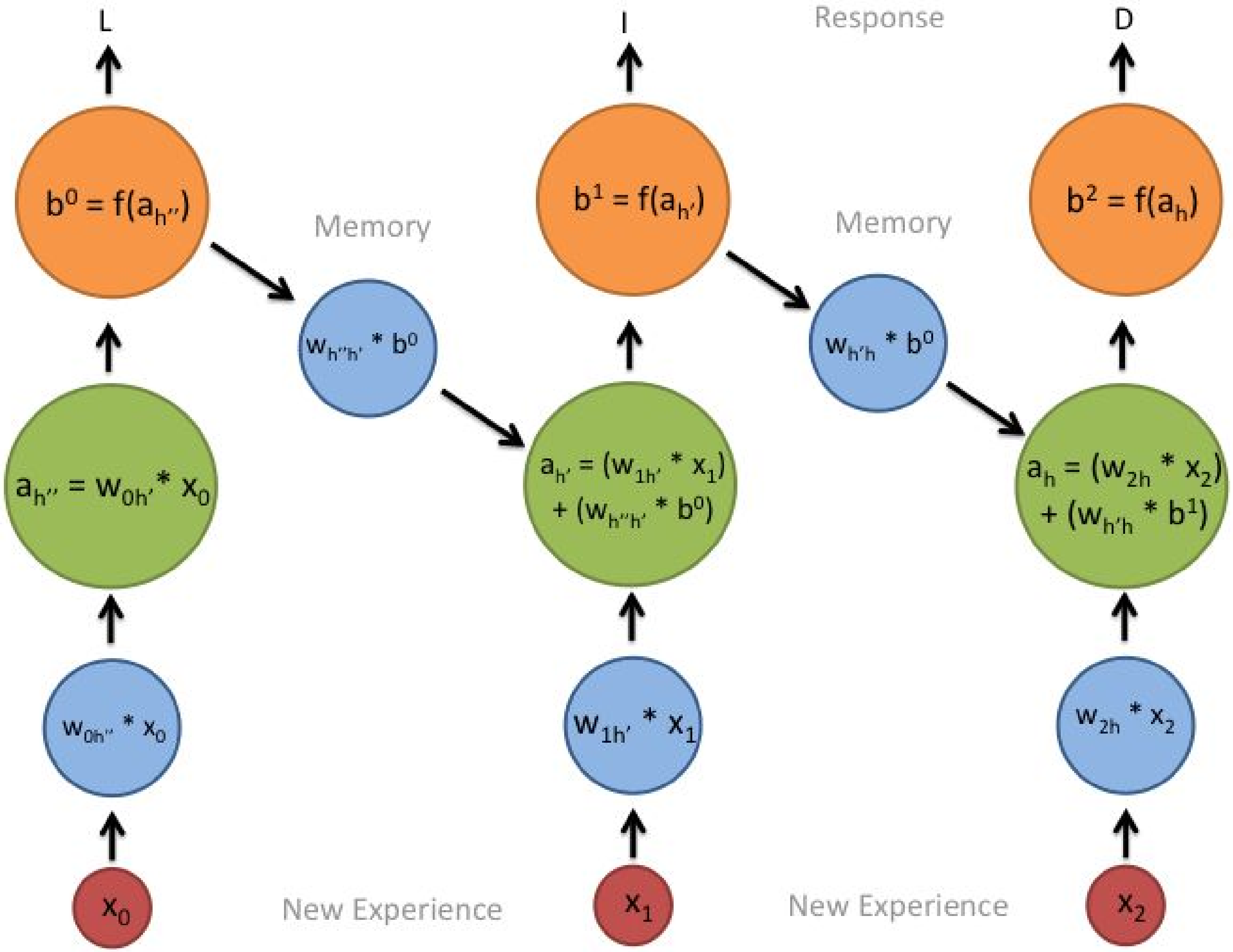
Some issues that needed to be resolved for the network to be useful:

1. How to train the network with Back-propagation.

2. How to stop gradients vanishing or exploding during training.

Recurrent neural network



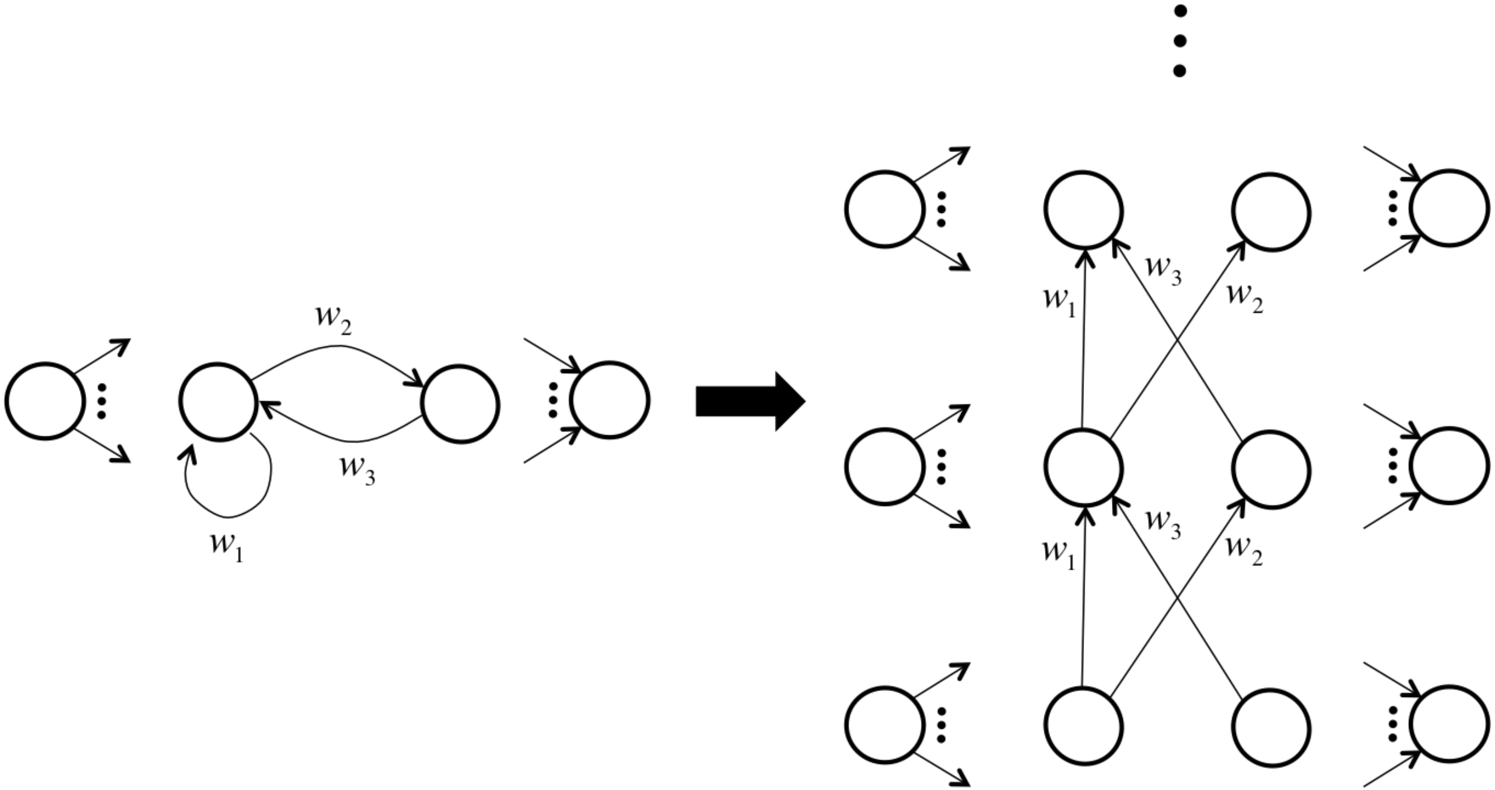


https://imgur.com/6Uak4vF

https://deeplearning4j.org/lstm

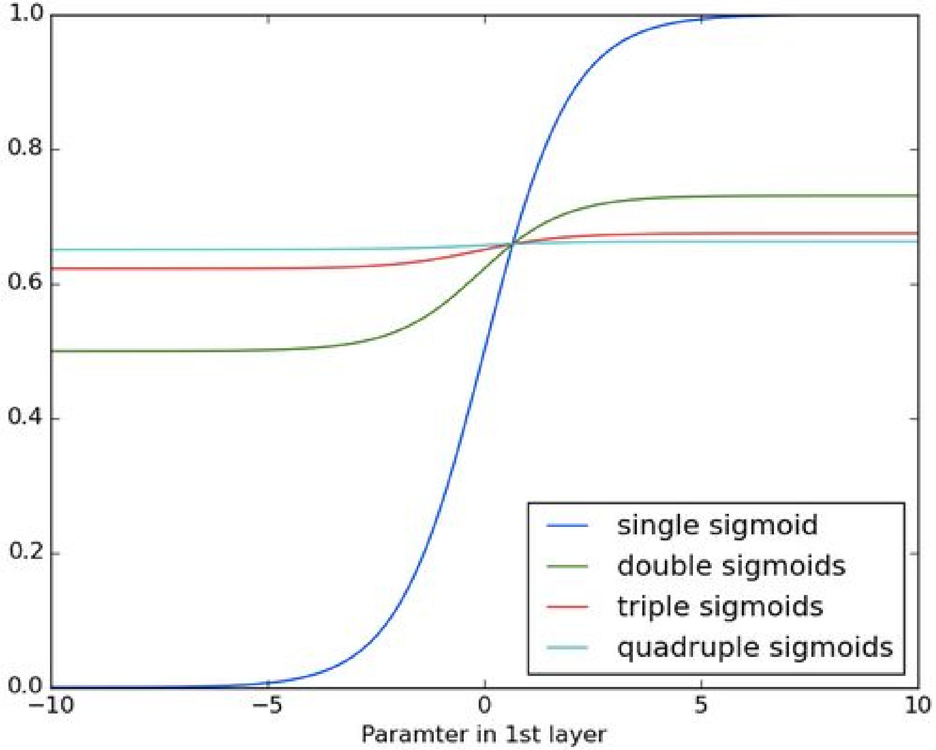
DATA SCIENCE RETREAT®

Unrolling



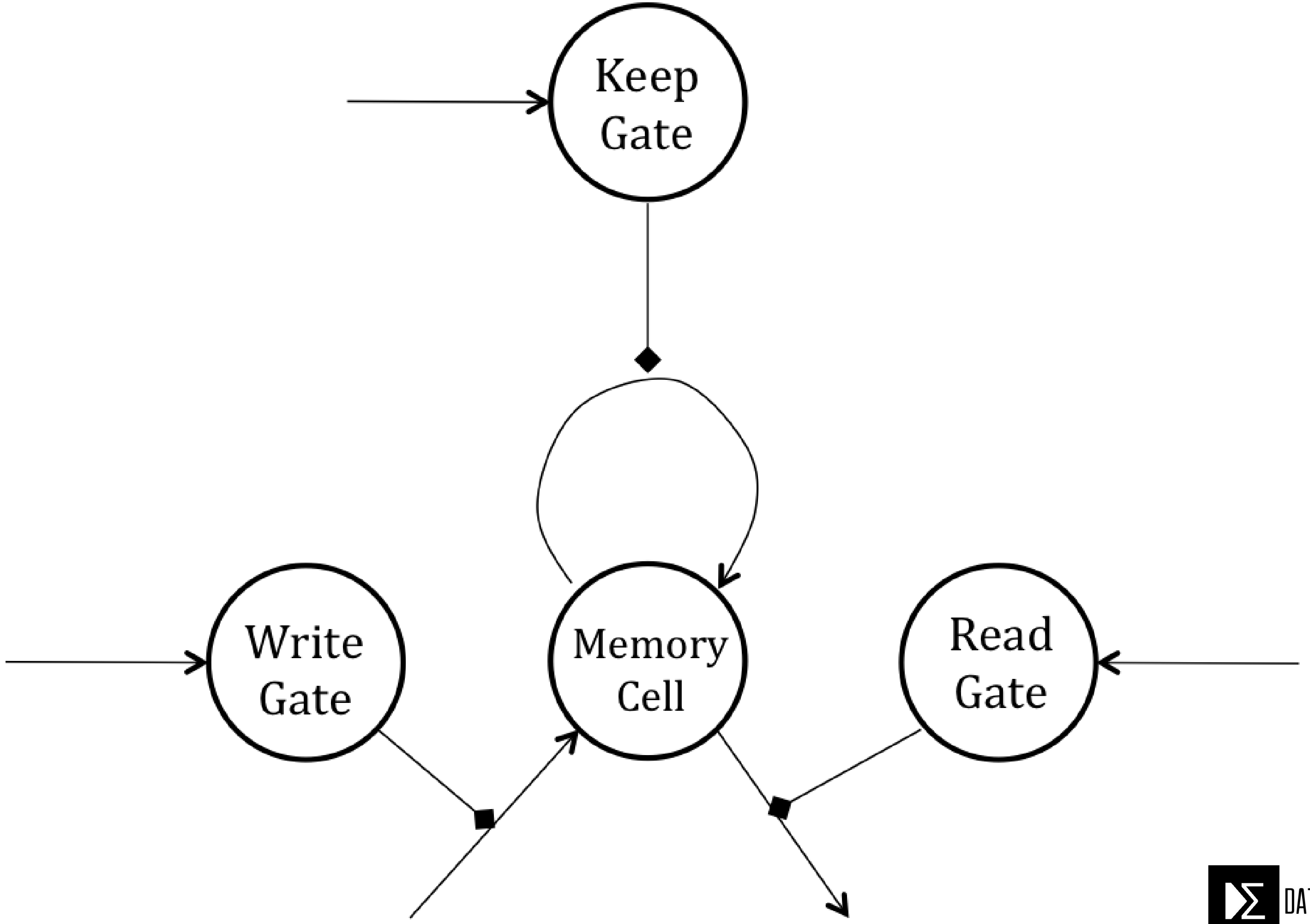
DATA SCIENCE RETREAT®

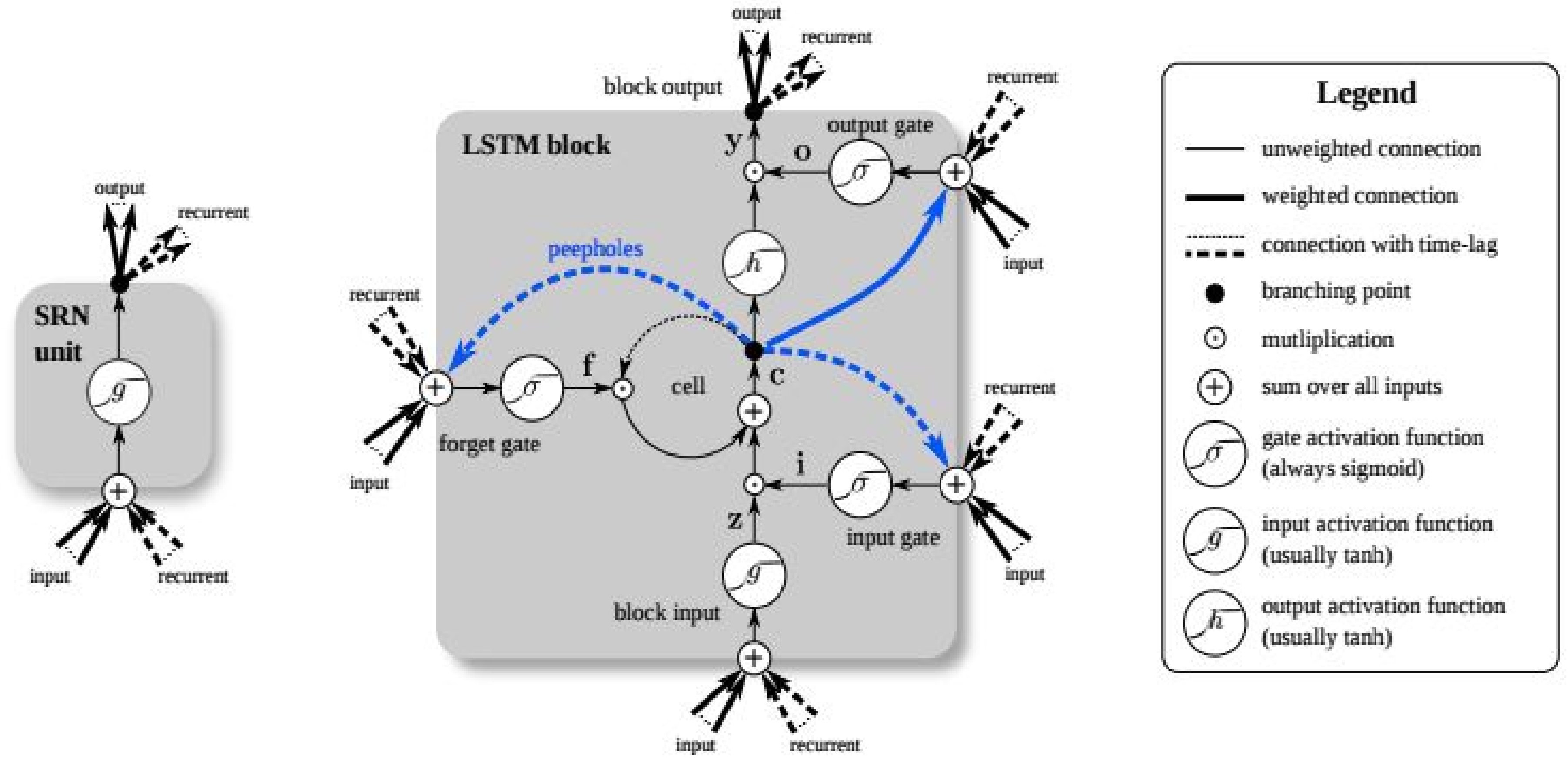
Vanishing (and Exploding) Gradients



DATA SCIENCE RETREAT®

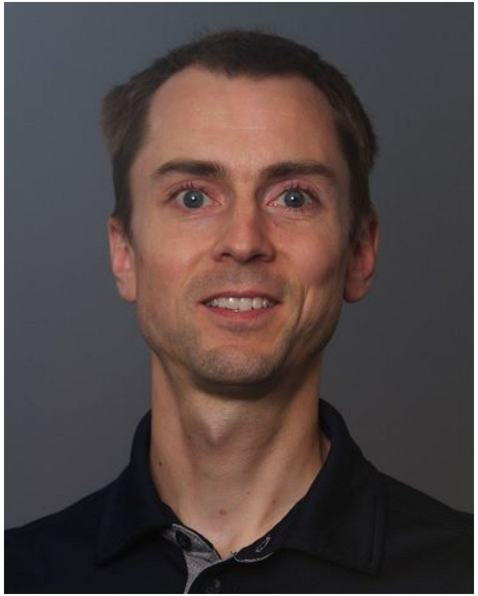
LSTMs contain information outside the normal flow of the recurrent network in a gated cell. Information can be stored in, written to, or read from a cell, much like data in a computer’s memory


Keep
Gate
Write
Gate
Memory
Cell
Read
Gate



re *1*. Detailed schematic of the Simple Recurrent Network (SRN) unit (left) and a Long Short-Term Memory block (right) as used
e hidden layers of a recurrent neural network.

End-to-end LSTM-based dialog control optimized with supervised and reinforcement learning



Jason D. Williams and Geoffrey Zweig
Microsoft Research

DATA SCIENCE RETREAT®

How can I help you?

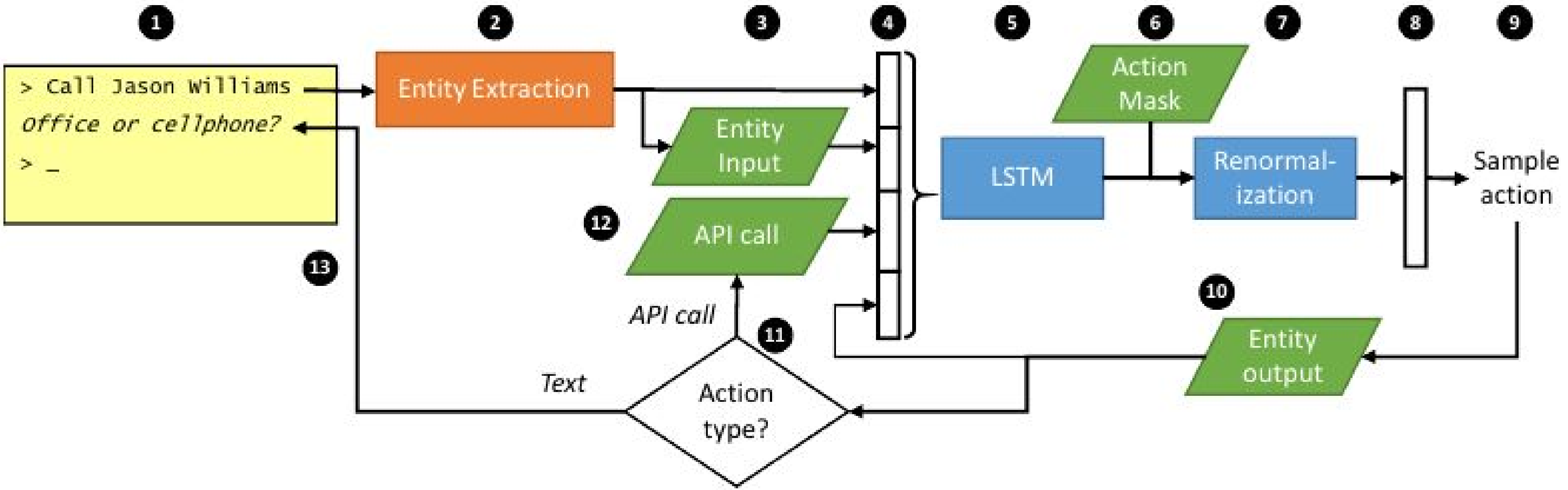
Call Jason

Which type of phone: mobile or work?

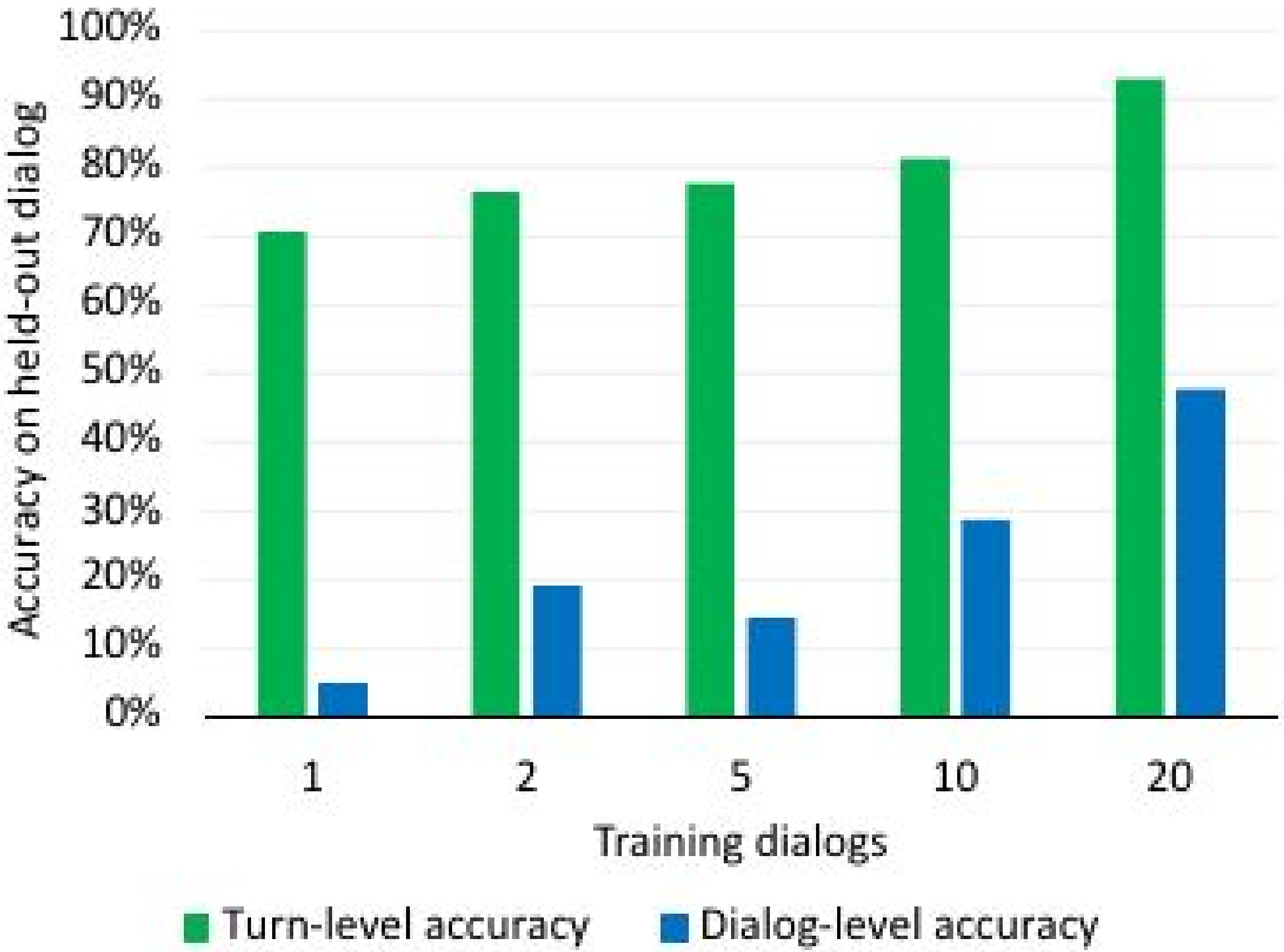
Oh, actually call Mike on his office phone

Calling Michael Seltzer, work.

PlaceCall



DATA SCIENCE RETREAT®



100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Accuracy on held-out dialog
1
2
5
10
20
Training dialogs
Turn-level accuracy
Dialog-level accuracy

Results

When to use

Use the simplest possible model first

When to use

As a rule of thumb, SVMs are great for relatively small data sets with fewer outliers. Random forests may require more data but they almost always come up with a pretty robust model.

Deep learning algorithms require "relatively" large datasets to work well, and you also need the infrastructure to train them in reasonable time.

Deep learning algorithms require much more experience: Setting up a neural network using deep learning algorithms is much more tedious than using an off-the-shelf classifiers such as random forests and SVM

Pros and cons

PROS

- No feature engineering necessary
- Setting the state of the art for
  - computer vision
  - speech recognition and
  - some text analysis
- Full potential still unknown
- They get better with more data

CONS

- Doesn't work out of the box
  - Many choices
  - Architecture
  - Hyper parameters
  - learning algorithm
- Needs huge amounts of data
- Computationally expensive

➡ lots of expensive experiments

Can we get rid of traffic lights?

SCIENCE RETREAT®